TAGGING TEMPORAL EVENTS IN TEXT

A CSU44061 Project by Group 20

Decemeber 24 2020

Mohamed Suliman
Student No: 17327728

Oskar Cahill
Student No: 17327771

Con Óg Ó Laoghaire
Student No: 16340036

The source code is available at:
https://github.com/namilus/temporal-event-tagger

Disclaimer:

This project is different to that proposed originally. In the original proposal, we intended to tag the temporal relations between events as well as the events themselves. In this report we only undertake tagging temporal events. Professor Doug Leith was emailed and this change in project scope was approved. Also note that excluding this title page and the contributions, this report fits the page limit of 5 pages.

1 INTRODUCTION (M.S & C.o.O.L)

Rarely does one find some text, be it a news article, blog, chapter of a book, tweet, etc., that does not contain some degree of temporal information. The majority of texts we read are full of events that occur, either in the past, in the future, or are happening right now. The extraction of these temporal events from natural language is beneficial to many applications in computer science, Question Answering (QA) and text summarization to name a few. In this report we describe our approach in extracting temporal events from texts. Examples of temporal events are given below:

- "John *answered* the telephone"
- "An *investigation* was undertaken"

In these examples, the events are "answered" and "investigation". Typically we find that tensed verbs and sometimes nouns, depending on the context, are events. Our goal here is to predict whether each word in a block of text, be it a news article, blog, etc., represents an event or not.

Another way to think about this problem is that we are given a list of words, and we need to determine which of these words are an event which are not an event. What we have is a classification problem. However here we are dealing with words, which is unlike most machine learning applications, where we deal with an n-length vector of numbers. Established techniques of vectorizing text operate at the document level *e.g.* vectorizing a document that has already been labelled for its sentiment. Models like `word2vec` are concerned with converting a word into a vector, so that words with similiar meanings are nearby in the vector space. However, a word being tagged as an event also depends on the sequence of words that came before it. Depending on its context, a word will be tagged as an event, *e.g* "The stock price *rose* yesterday", or not an event, *e.g* "He gave his wife a red *rose*". We take context into account by using a sequence classifier called a Hidden Markov Model (HMM). Section 2 explains our training set, the TimeBank 1.2 corpus, and the features we extracted. Section 3 describes how we trained our HMM. Section 4 describes and discusses the evaluation of our model. Finally in Section 5 we conclude our findings.

2 DATASET & FEATURES (M.S & C.o.O.L & O.C)

The dataset we chose is the Timebank 1.2 Corpus, available from the Linguistic Data Consortium (LDC). This is a collection of 183 news articles spanning a range of topics from corporate reporting and financial markets, to politics and crime. Each file can range from several sentences to several paragraphs in length. What is special about these news articles is that they are annotated with an XML Markup Language called TimeML. The TimeML specification allows for temporal events to be tagged. Any text within the `<EVENT></EVENT>` tags is said to represent an event; this makes for an excellent training set for our model, as events are already labelled. Choosing our own set of documents–be they news articles, blogs, and etc–and labelling the events ourselves would have been frought with errors in linguistically determining what constitutes an event. The TimeBank corpus is used here as it has been annotated by three expert annotators and for the `EVENT` tags, a high inter-annotator agreement[1] of 0.78 is reported. Thus we can safely assume that the annotations of events in the TimeBank corpus are correct due to the high level of agreement between the annotators.

2.1 FEATURES FOR HIDDEN MARKOV MODEL (C.o.O.L & O.C)

Although the operation of the HMM is described in further detail in Section 3, we will describe some details here in order to fully explain the justification for our feature choices.

Two constituents of a HMM include the set of symbols and the set of states. The symbols are the words that we observe as part of the text–which can be thought of as a function of the state the HMM is currently in. As we go through each word in the document, it is thought that the model goes from state to state and that the state it is currently in says something about the current word; *i.e* our model could have some state in it and whenever the model is in that state, then we can say the current word or *emission*, is an event. The transitions from state to state are said to be "hidden", and it is these transitions we wish to elucidate with our model.

The idea of tuning hyperparameters does not translate well to HMMs. There are no numerical features of which we can get a polynomial of some degree from, and no regularization parameter. Thus apart from altering the size of the training set, the only things we can change around are the two sets of symbols and states.

[1] Inter annotator agreement is a measure of how different annotators agree when annotating a document. This value can be between 0 and 1, and it is understood that a higher value means that there is a high level of agreement by the annotators, increasing confidence on the true value of the annotation.

For the set of states, this seems clear. Words are either events or they are not. Thus we have two states: an event state and a non-event state. When choosing the set of symbols, we can set it to be the vocabulary of our dataset. How we deal with words that may appear, which are not part of this vocabulary, is explained further in the Section 4.1.1.

As mentioned previously, we find that tensed verbs (that is, verbs which are not in the infinitive form) are often tagged as events. Thus, instead of using the 'raw' word, we can replace the sequence of words in the document with the sequence of their POS tags. The NLP library `nltk` provides functionality to determine the Part-of-Speech (POS) tag of a word[2]. It is clear that whether a word is a verb, noun, preposition, or etc, plays a large part on whether it is labelled as an event. For example, in a document we may find a sentence that reads:

> The stock price dropped dramatically on Thursday after the CEO resigned.

We can replace this with its POS tag sequence, which would be:

> DT NN NN VBD RB IN NN IN DT NN VBD

NLTK uses the Penn Treebank POS Tagset, and we will use this as our set of symbols, comparing how this approach behaves to just using the 'raw' words themselves.

3 METHODS (M.S)

We decided to use a Hidden Markov Model to label words. Text is, at its core, a sequence. Therefore sequence classifiers like HMMs are the best tool for the job. We define a HMM $\lambda = (A, B)$, where A is the transition matrix and B is the observation likelihood matrix[3]. Each a_{ij} in A represents the transition probability $P(s_j|s_i)$ going from state s_i to state s_j. This is known as the *Markov assumption*, which means that the next state is determined by only the current state of the model. B represents the probability of a word being emitted given the current state, $P(w_i|s_i)$. These probabilities are computed using the Maximum Likelihood Estimate (MLE) :

$$P(s_i|s_{i-1}) = \frac{C(s_{i-1}, s_i)}{C(s_{i-1})}$$

$$P(w_i|s_i) = \frac{C(w_i, s_i)}{C(s_i)}$$

[2]This was also trained using a HMM and acheives 97% accuracy.

[3]This notation is taken from Jurafsky & Martin

When calculating the transition probabilites $P(s_i|s_{i-1})$, we count the amount of times a word tagged with s_i is preceded by a word tagged s_{i-1} and then divide by the total times the s_{i-1} tag has been observed in the training data; these are just bigram counts that are used in NLP to create probabilistic language models. Emission probabilities $P(w_i|s_i)$ are calculated by dividing the total amount of times emission w_i is tagged as s_i, by the total occurences of s_i in the training data.

Figure 1 shows the HMM that we have come up with. We have 4 states: `<s>` represents the start of a sentence, `E` is the event state, `N` is the non-event state, and `</s>` represents the end of a sentence. We feed into the HMM a sentence and it will tag words in that sentence - hence we need two extra states to denote the start and end. The arcs represent the possible transitions, and they are usually labelled by the probability of that transition happening. HMMs also take another parameter, π, which is the probability distribution of which state we start in. In this case, we always start at state `<s>`.

If we have a sequence of n words, w_1^n, and we want the corresponding tag sequence, s_1^n, we find this by calculating the most probable sequence. We have that $\hat{s}_1^n$, the most probable sequence of tags, is:

$$\begin{aligned} \hat{s}_1^n &= \underset{s_1^n}{\operatorname{argmax}} P(s_1^n|w_1^n) \\ &\approx \underset{s_1^n}{\operatorname{argmax}} \prod_{i=1}^{n} P(w_i|s_i)P(s_i|s_{i-1}) \end{aligned}$$

The Viterbi algorithm is a dynamic programming algorithm that is used to find the sequence of tags that maximises $P(w_i|s_i)P(s_i|s_{i-1})$. We used `nltk`'s implementation of this algorithm in our project.

3.1 PIPELINE (M.S)

In this section we will describe how we take the TimeML annotated news article and generate labelled sentences with which we train our HMM, as well as the format of the outputs generated. Firstly, we parse the XML and extract the raw text of the document, sentence by sentence, keeping track of which words have been annotated as events. We then calculate the transition and emission probabilities based on the training set. From this we use the `nltk.tag` module to create the Hidden Markov Model from these probabilities. This module takes care of calculating the most probable sequence of tags using the Viterbi algorithm. The full procedure is as follows:

- After parsing the XML, we have the sentence: "The stock price dropped dramatically on Thursday following the CEO's resignation"

- Then we tokenize, remove capital letters, and add markers to denote the beginning and end of the sentence:

  START the stock price dropped dramatically on thursday after the ceo resigned END

- From here we can choose to convert the words to their corresponding POS tags:

  START DT NN NN VBD RB IN NN IN DT NN VBD END

- These sentences are then used to calculate the transition probabilities and observation likelihoods. After determining the most probable tag sequence, a possible output from the HMM would look something like the following:

  <s> N N N E N N N N N N E </s>

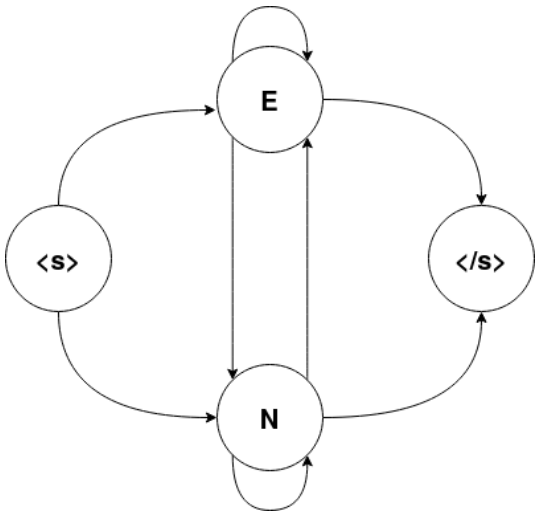

Figure 1: The HMM we employ.

4 EXPERIMENTS (M.S & O.C & C.o.O.L)

We run several experiments here to evaluate the peformance of our model. First we look at how the model behaves when we just train it on the raw emission words *i.e* there is no conversion of the words in the document to their POS tags. Then we will see how much better or worse, if any, the model performs when we do this conversion, as well as its comparison to some baseline models. The motivation for conversion into POS tags is two-fold. Firstly, when using just the raw words, we are very likely to encounter words that our model has not been trained on. We take care of this using smoothing techniques that give psuedocounts for out-of-vocabulary (OOV) words so that the probabilities are not zero, but something more aligned with reality. We discuss the different types of smoothing and their effects on performance below. When smoothing, the probabilities in the observation likelihood matrix need to be recalculated every time we encounter an OOV word thus making testing a lot more computationally demanding. The idea behind replacing words with their POS is that the vocabulary is much smaller (~50 symbols) and so this drastically reduces the chance of an OOV words and we have dont have to do as much, if any, smoothing of probabilities. Secondly, as mentioned above, we see that tensed verbs are almost always tagged as events *e.g* "I *went* to the store", "He *dialled* the number", etc., and so maybe there is a trend here to be learned that is captured best through parts of speech.

4.1 RESULTS & DISCUSSION (M.S & O.C & C.o.O.L)

We ran 5-fold cross validation on 2 models, one trained on the raw emissions and the other trained only on the POS tags. Table 1 shows how the two models performed when predicting the data they were trained on. We can see for both, accuracy is very high, and there is a high disparity between the raw model's f_1 score and that of the POS model. Table 2 shows the same metrics but for the 2 models' predictions on an unseen test set. Looking at accuracy, it stays around the same as the training accuracy, with a drop of .04 for the model trained on the raw emissions. What is clear though is that the raw model performs much better than the POS model, showing that only part-of-speech is not sufficient to train a HMM that can accurately, and precisely tag temporal events in text.

The f_1 score on the test set for the 2 models is interesting here. The raw model's f_1 score decreases for the test set compared to the training set. This is due to the OOV words in the test set. Our model has never seen these words in training, and we don't see the tag for this word *i.e* we dont know whether it is an event or not, we just assume it's not an event, given that the majority of words in the training set are not events and smooth the probabilities accordingly. This has clear implications on the models performance, as we can the f_1 score drops by 0.16. Section 4.1.1 tests other methods of smoothing and looks how close each gets to the training set f_1 score.

We can also see that despite both models having a very high accuracy, their f_1 scores are comparatively much lower. This can only be due to the proportion of event and non-event words in the texts. Events are rare, and non-events make up the majority. Thus true negatives make up a large

part of the confusion matrix and skew the accuracy closer to the optimal value of 1.

Tables 3 and 4 shows the confusion matrices for both the raw and POS models after having a train test split of 80:20 and tested on the unseen test set. We can clearly see that there a lot of true negatives which skew the accuracy. Therefore the accuracy of the model is not representative of its performance, and it is for that reason we include the f_1 score. This combines precision and recall into one metric and gives us a much better idea of how or models are performing.

Interestingly, the metrics for the POS model don't change going from the training set to the test set. One could look at the model trained on the raw emissions and argue it is overfitted to the data. However this is just because the size of the vocabulary in the training set is relatively small and that OOV are encountered often. If this training set had a more comprehensive vocabulary, we wouldn't see such a disparity between the raw model's training metrics and it's test metrics.

Table 1: Evaluation metrics for our model on the training set.

model	accuracy	f_1 score
raw	$(0.9615, \pm 0.0005)$	$(0.8412, \pm 0.0021)$
POS	$(0.8940, \pm 0.0014)$	$(0.4229, \pm 0.0165)$

Table 2: Evaluation metrics for our model on the test set.

model	accuracy	f_1 score
raw	$(0.9291, \pm 0.0011)$	$(0.6691, \pm 0.0192)$
POS	$(0.8936, \pm 0.0040)$	$(0.4219, \pm 0.0337)$

Table 3: The confusion matrix of the raw model.

TN	FP	FN	TP
9897	237	542	914

4.1.1 SMOOTHING (M.S)

The previous models trained on the raw emissions were assuming every OOV word to be a non-event. Precisely, what this means is that we apply add-1 (laplace) smoothing to the observation likelihood probabilities. Given an unseen word w', then $P(w'|N) = \frac{1}{C(N)}$. Now since $C(s)$ where s is the non-event state has increased by 1, we also need to update the probabilities for every other word in our vocabulary. This is of course time intensive and is one of the motivaitions for converting the raw emissions into their POS tags.

Clearly this isn't the best way to deal with OOV words. Instead of always assuming the word is not an event, another technique takes advantage of the idea expressed in previous sections, in that tensed verbs are almost always tagged as events. The new smoothing technique is as follows:

Table 4: The confusion matrix of the model trained on POS tags.

TN	FP	FN	TP
9859	247	915	555

- If the OOV word is a tensed verb, update the probabilities assuming it is an event
- Else, it is a non-event word, and update the probabilities accordingly

Again with this new method of smoothing, we run 5-fold cross validation, and the results are shown in Table 5. The f_1 score for the test set jumps significantly, from the 0.6691 reported with the old smoothing to 0.7115 when using this new smoothing technique. Accuracy stays around the same value, but what we see here is a marked increase in performance from what is at most a heuristic about the data. Figure 2 shows the difference in f_1 scores for the two different smoothing techniques. Indeed it is not only verbs that can be events, and any further improvement in performance will have to take that into account. We do not do so here.

Table 5: 5-fold cross validation results with new smoothing technique

set	accuracy	f_1 score
train	$(0.9615, \pm 0.0004)$	$(0.8412, \pm 0.0021)$
test	$(0.9339, \pm 0.0015)$	$(0.7115, \pm 0.0153)$

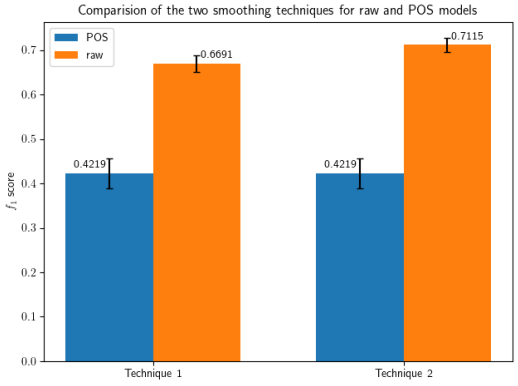

Figure 2: The improvement in f_1 after applying the new smoothing technique.

4.1.2 BASELINE COMPARISONS (M.S & O.C & C.o.O.L)

A baseline that we can use here is the idea mentioned throughout this report. Tensed verbs are often events. Therefore we can train a baseline to tag words as events if they are a verb. What we want to see how much better our HMM model with all of its transition probabilities and observation likelihoods performs compared to this simple heuristic. We did an 80:20 split and tested this baseline on the 20% test set so the numbers in the confusion matrix are similiar to those shown previously.

Table 6: The accuracy and f_1 of our baseline.

model	accuracy	f_1 score
baseline	$(0.8757, \pm 0.0832)$	$(0.4651, \pm 0.2731)$

Table 7: The confusion matrix for the baseline.

TN	FP	FN	TP
9537	534	856	608

Table 6 shows the accuracy and more importantly the baseline's f_1 score. It is much worse than our trained model, with a mean of 0.4651 for each sentence and a very large standard error. The baseline also makes more errors in false positives and false negatives than our model. It is clear then that just this heuristic about verbs being events is not sufficient to create a precise tagger.

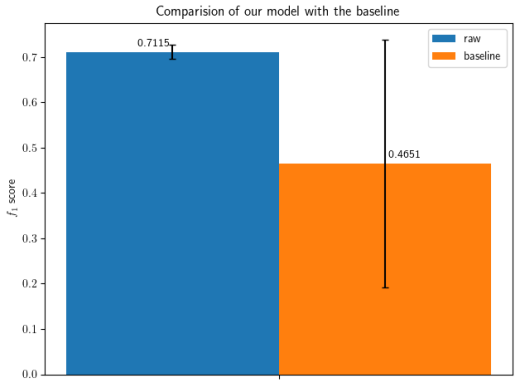

Figure 3: Comparison of our raw model's f_1 to that of the baseline.

5 SUMMARY (M.S & C.o.O.L)

A group of researchers working on TimeML created their own model that tags temporal events. This model is part of their TARSQI[4] system. They report an f_1 score of 0.8012. While we have not acheived results like these, we have developed a model that can tag events with some degree of accuracy and precision. Initially, we had thought the scope of this project would be broader. However, as we progressed with development it became apparent to us that a handful of factors, namely time taken to create a working tagger and our limited knowledge about linguistics, forced us to narrow the scope. Most of the work went into processing the data. The TimeML XML Schema took some time to learn and to parse accurately. It is clear though that a HMM model is very good at classifying sequences of words, and replacing words with their POS tags, while offers some benefit in terms of computation time, gets rid of a lot of the detail that the raw words provide our model.

This project was certainly an engaging task and we feel that we have provided our best effort at creating a method for extracting events from texts.

M.S, O.C, C.o.O.L

[4]Temporal Awareness and Reasoning Systems for Question Interpretation

6 CODE CONTRIBUTIONS

This section includes who contributed to which source code files:

- `src/scripts/relabel.py` M.S
- `src/group20/experiments.py` M.S & O.C
- `src/group20/__main__.py` M.S & C.o.O.L
- `src/group20/model/model.py` M.S
- `src/group20/model/validator.py` M.S & O.C & C.o.O.L
- `src/group20/plots.py` M.S & C.o.O.L
- `src/group20/timeml/timemldoc.py` M.S
- `src/README.org` M.S